Computer Vision - Assignment 1 - v0.3

Contact:

- Rui Nóbrega: Discord or email (<rui.nobrega@fct.unl.pt> Subject: [CV] - topic)

Changelog

10/Oct - Warning about sorting files!!! Section 3.
10/Oct - evaluation points added. Minor clarifications.
8/Oct - added Part 2 with sections 6, 7 and Summary in the end.

1- Introduction

Alice and Bob went on holiday and took a large quantity of photos. As always, there is a large number of repeated photos or sequences of photos from the same place. We want to write a program that analyses the picture set and divides the photos into groups. This division will use the techniques learned in the previous labs focusing especially on color analysis and feature analysis. In this assignment try to use these techniques to perform as best as they can, knowing that a 100% accuracy result might not be possible. The assignment will use a small subset of the INRIA holidays dataset (https://www.kaggle.com/datasets/vadimshabashov/inria-holidays).

2- Rules and Description

- The assignment will be done in Groups of 2 (except students with student-worker status in CLIP), the report and python files should be identified with name and number. Single groups will only be accepted if explicitly authorized by the professor by email.
- All the python code should be in the root of the project(e.g., `50000_60000` folder). The work will be evaluated using the starting code `python tp1.py`.
- In moodle, there is a starting code. This is the **mandatory** structure of the code:

```
.
├── input                #you can only read these files, you cannot write them
│   └── groundtruth.json
│   └── images_in.jpg
├── output
│   └── similar-0
│   │   └── images_out.jpg
│   └── images_out.jpg
├── 50000_60000      #change to your student numbers,
    |                #smaller number first,
    |                #this is what you deliver.
    ├── report.ipynb
    └── tp1.py
    └── additional_files.py
```

- No external libraries are allowed, only libraries used in the labs, namely `OpenCV, Numpy or PyPlot` or standard python/JSON input/output libraries.
- The use of AI for reasons other than consulting what a certain function does or sentence/word correction should be declared in a separate section in the report.
- The students are obliged to follow the code of ethics of the department of computer science.

3- Preparing the images

All the operations will be performed in a resized version of the images.

- Resize every image in the `input` folder maintaining the aspect-ratio of the images. The smaller side of each image should be 512 pixels.
- You don't need to save the images, depends on the operations of the next steps.
- DO NOT WRITE THE IMAGES IN THE INPUT FOLDER (this folder will be read only in the evaluation). If you need to temporarily write the images use the `output` folder.
- **IMPORTANT:** please do a sort in the names of the images that you will open so that it works correctly across all file systems (Windows,Mac and Linux). Example:
  `# Get all image files (jpg, png, jpeg)`
  `image_paths = sorted(glob.glob(os.path.join(input_dir, "*.*")))`

4- Finding similar images

Using color analysis and the histogram of each image find the groups of images that are more similar.

- Create a distance function that compares two histograms and gives a score on how distant they are. You can use the `Bhattacharyya` or the `Chi-Square` distance to compute the difference between two histograms. Implement yourself a function called `histogram_distance` that does this. Describe it in the final report.
- Each time there is an image that has more than 2 other images that are similar, create a folder and write the original image (resized with the original name) and the similar images in that folder. The folder names should be `similar-#`, with `#` being a sequential number (*e.g.,* similar-0, similar-1, ...).
- Assume that the pictures were taken in chronological sequence and that if an image is too different from the previous there is no need to check the next ones. This means that you only should need to test until you find a dissimilar image.

5- Compute common histogram and equalize the white balance of the photos

- For each folder `similar` find the average histogram. Save an image `histograms.jpg` with the final histogram. *Extra points* if you print the histograms of all images together with the average histogram with labels.
- Perform a white balance on the images in that folder centered around the average from the average histogram calculated before.

6- Create your own descriptor based on MOPS (4 points)

Create functions to support your own descriptor inspired on MOPS (see lectures). You may either improve on what is suggested or implement only certain parts (with decreased grade).

- `my_track_points`: Function that find points using `cv2.goodFeaturesToTrack`.
- `my_point_rotation`: Function that given a point and an area around the point, finds its rotation (use `cv2.Sobel, $\theta = arctan(Iy, Ix)$ and histogram to find dominant angle)
- `my_descriptor`: Function that creates your own descriptor for each point.
  - The goal is to take a window around the point (e.g.,40x40), rotate the image and downsample it to 8x8. The result is a flatten array of 64 float values normalized

between 0 and 1. Explore functions such as `cv2.getRotationMatrix2D` and `cv2.warpAffine` and try to create the best descriptor you can. Present and explain the algorithm and images of the 8x8 patch in the report.

- `my_distance` : Create a function that given two descriptors gives a match distance score using the Euclidean Distance.
- `my_match` : Create a function that gives a match based on the Nearest Neighbours ratio and a certain Threshold.
- output an image comparison ("my_match.jpg") in the `output` directory using your algorithm and using SIFT (e.g., Use these two "109900.jpg" "109901.jpg")

It's up to the students to decide the arguments and return values of each function. In the report present the match between two images using your descriptor and using SIFT (e.g., Use these two "109900.jpg" "109901.jpg"), comment on the results namely what is the average matching distance with your descriptor compared with the SIFT descriptor.

7- Compare all images and print stats

Using SIFT (or your MOPS descriptor if you are confident enough):

- Find very similar photos :
  - for every similar folder, compare each image with the first image of the folder and verify if it is identical. If it is, save the keypoint comparison in a file named `equal-0.jpg, equal-1.jpg, etc' in the same folder.

For each `similar` folder print the following output:
`similar-{a} number of images: {b} ground-truth: {c} precision: {d} averagecolor: {e}`

```
* {a} is the sequential number of similar.
* {b} is the number of images in that folder.
* {c} is the number of correct matches according to the `groundtruth.json`.
* {d} is the precision with 3 decimal points $precision=abs(b-c)/c$
* {e} is the RGB value of the average color of that folder(before white
    balance)
```

- in the end, there should be a global count for all the folders
  `TOTAL number of images: {b} ground-truth: {c} precision: {d}`

8- Report (7 points)

- Create an interactive report in the form of a notebook that explains and exemplifies with code the steps 3 to 7, showing the results with images and console prints. Use two images to exemplify the algorithms.
- Provide insightful explanations of what you did on each step. You can present additional statistics and charts (e.g., histograms).
- Please identify your report.
- Explain any extras that you may have done in a separate section.

9- Evaluation

- You should submit a `zip` file of your numbered folder (e.g., 50000_60000.zip in our above example, no need for the input and output folders) in Moodle.
- Don't forget to use the best practices of programming. Present a clean, well organized code. You can separate the code in several files.
- The grade is from 0 to 20. If you complete everything that is asked you will get a grade from [0-18]. The grades 19 and 20 are reserved for outstanding assignments that excel and go above and beyond.
- The evaluation will be performed by using an alternative `input` set of images, using an empty `output` folder, running `python tp1.py` followed by running the `report.ipynb`. All elements delivered will be evaluated. Your code cannot write anything in the input folder. There will be a 2 minute timeout for the code to run.
- Deadlines: (Check Moodle).

Summary output:

- tp1.py output summary:
  - Several folders called `similar-#` in the `output` folder.
  - For each `similar-#:
    - white balanced images
    - histograms.jpg
    - equal-#.jpg
    - print in the console with stats
  - my_match.jpg
  - print final stats

- report.ipynb summary:
  - Introduction, description of sections 3 to 7, with live demos with two input images, Conclusion with self-evaluation.